**2022년도 졸업프로젝트[Back-end]**

**팀이름: 황제펭귄**

주제: TLS를 구현을 통한 홈페이지 제작 및 보안 채널 로그인 기능 구현

|  |  |  |  |
| --- | --- | --- | --- |
| **작성자** | 정경재 | **작성일자** | 2022.03.10 |

* **금일 진행 상황**

|  |  |
| --- | --- |
| **진행 상황** | **비고** |
| **질문 및 진행방식에 대한 논의**  **진행 중**   1. 스프링 부트와 AWS 로 혼자 구현하는 웹서비스 CHAPTER03 진행.   **완료**  스프링 부트에서 JPA로 데이터베이스 다뤄보자.  **JPA소개**  -데이터베이스를 다룰때 개발시간보다 SQL을 다루는 시간이 더 많아져 테이블 모델링에만 집중하게 되는 현상을 해결하기 위해 JPA라는 자바 표준 ORM기술 개발.[개발자가 객체지향적으로 개발을 하면 JPA가 관계형 데이터베이스에 맞게 SQL을 대신 생성 후 실행함]  -JPA는 Spring Data JPA 모듈을 이용해 다룸[구현체,저장소 교체의 용이성]  **프로젝트에 Spring Data JPA 적용하기**  **1.build.gradle(의존성)**  dependencies **{//** compile이 인식되지 않아 implementation으로 교체하였음implementation 'com.h2database:h2'  implementation 'org.springframework.boot:spring-boot-starter-web'  implementation'org.projectlombok:lombok'  implementation'org.springframework.boot:spring-boot-starter-data-jpa'  implementation'com.h2database:h2'//h2 데이터베이스 커넥션을 위해 추가함.  annotationProcessor('org.projectlombok:lombok')  testAnnotationProcessor('org.projectlombok:lombok')// 테스트가 작동하지 않아 추가하였음 롬북을 사용할때 필수적으로 추가하여야한다고 함.   testImplementation'org.springframework.boot:spring-boot-starter-test' **}**  **2.posts 클래스:실제 DB의 테이블과 매칭될 클래스이다.**  **-domain 하위에 posts패키지 하위에 생성**  package com.example.study2.domain.posts;  import lombok.Builder; import lombok.Getter; import lombok.NoArgsConstructor;  import javax.persistence.Column; import javax.persistence.Entity; import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Id;  @Getter//클래스 내 필드의 Getter 메소드 자동생성 @NoArgsConstructor//기본 생성자 자동 추가 @Entity//테이블과 링크될 클래스.ex)SalesManager.java-sales\_manager table과 같이 클래스의 카멜케이스 이름을 언더스코어 네이밍 테이블과 매칭시킨다.  public class Posts {  @Id  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  private Long id;  @Column(length = 500, nullable = false)  private String title;  @Column(columnDefinition = "TEXT", nullable = false)  private String content;  private String author;   @Builder//해당 클래스의 빌더패턴 클래스 생성,생성자 상단에 선언시 생성자에 포함된 필드만 빌더에 포함.  public Posts(String title, String content, String author) {  this.title = title;  this.content = content;  this.author = author;   }  #Getter와@NoArgsConstructor은 롬북의 어노테이션, @Entity는 JPA의 어노테이션.  #Getter/Setter무작위 생성시 차후 기능변경이 필요할 때 복잡해지므로 Entity클래스에서는 Setter메소드를 만들지 않고 빌더 클래스를 사용해 DB를 채움. **2. postsRepository 인터페이스: Posts클래스롤 Database에 접근하게 해줌[DB Layer 접근자]**  **#Entity클래스와 Entity Repository는 Entity클래스가 기본 Repository없이는 제대로 역할을 수행할 수 없어 함께 위치해야 함.**  **-domain 하위에 posts패키지 하위에 생성**  package com.example.study2.domain.posts;  import org.springframework.data.jpa.repository.JpaRepository;  public interface PostsRepository extends JpaRepository<Posts,Long>{ }  **3.Spring Data JPA 테스트 코드**  package com.example.study2.domain.posts;  import org.hibernate.type.TrueFalseType; import org.junit.jupiter.api.AfterEach;//After가 AfterEach로 변경되었다. import org.junit.jupiter.api.Test; import org.junit.jupiter.api.extension.ExtendWith;//RunWith가 ExtendWith로 변경 되었다. import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.test.context.SpringBootTest; import org.springframework.test.context.junit.jupiter.SpringExtension;   import java.util.List; import static org.assertj.core.api.Assertions.*assertThat*;  @ExtendWith(SpringExtension.class) @SpringBootTest public class PostsRepositoryTest {   @Autowired  PostsRepository postsRepository;   @AfterEach//단위 테스트가 끝날때마다 수행되는 메소드를 지정한다.여러 테스트 가 동시에 진행되면 다음테스트 실행시 실패할 수 있으므로 데이터 침범을 막기 위해 사용함.  public void cleanup(){   postsRepository.deleteAll();  }   @Test  public void 게시글저장\_불러오기() {  String title = "테스트 게시글";  String content ="테스트 본문";   postsRepository.save(Posts.*builder*()//.save는 테이블 posts에 insert/update 쿼리를 실행한다. Id값이 존재하면 update 없다면 insert쿼리를 실행함.  .title(title)  .content(content)  .author("jojodu@gmail.com")  .build());   List<Posts> postsList = postsRepository.findAll();//테이블의 모든 데이터를 조회해온다.   Posts posts= postsList.get(0);  *assertThat*(posts.getTitle()).isEqualTo(title);  *assertThat*(posts.getContent()).isEqualTo(content);  } }  **#** annotationProcessor('org.projectlombok:lombok')  testAnnotationProcessor('org.projectlombok:lombok')을 build.gradle에 추가하지 않으면 save,findALL,assertThat이 Cannot find symbol오류가 발생한다.  **의문사항**  **메인 어플리케이션 수행 시 warn 메세지:false값으로 바꿔주면 뜨지 않는다고 하는데 spring.jpa.open-in-view가 read파일에 들어있어 변경이 불가능하다.**  **2022-03-10 05:32:56.523 WARN 18584 --- [ main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning**  **4. 등록 수정조회를 위한 API만들기 3개의 Class가 필요하다. 1. Request 데이터를 받을 Dto 2. API의 요청을 받을 Controller 3. 트랜젝션, 도메인 기능간의 순서를 보장하는 Service.**  PostsAPiController  // 해당부분의 Packege명은 모두 다를 수 있음. package com.arad4228Leaned.arad4228.web;   import com.arad4228Leaned.arad4228.service.posts.PostsService; import com.arad4228Leaned.arad4228.web.dto.PostsSaveRequestDto; import lombok.RequiredArgsConstructor; import org.springframework.web.bind.annotation.PostMapping; import org.springframework.web.bind.annotation.RequestBody; import org.springframework.web.bind.annotation.RestController;  @RequiredArgsConstructor @RestController public class PostsApiController {   private final PostsService postsService;   @PostMapping("/api/v1/posts")  public Long save(@RequestBody PostsSaveRequestDto requestDto) {  return postsService.save(requestDto);  } }  PostsService  package com.arad4228Leaned.arad4228.service.posts;  import com.arad4228Leaned.arad4228.domain.posts.PostsRepository; import com.arad4228Leaned.arad4228.web.PostsApiController; import com.arad4228Leaned.arad4228.web.dto.PostsSaveRequestDto; import lombok.RequiredArgsConstructor; import org.springframework.stereotype.Service;  import javax.transaction.Transactional;  @RequiredArgsConstructor @Service public class PostsService {  private final PostsRepository postsRepository;   @Transactional  public Long save(PostsSaveRequestDto requestDto) {  return postsRepository.save(requestDto.toEntity()).getId();  } }  이부분에서 처럼 생성자를 AutoWire로 주입받는 것을 권장하지 않고 Bean에 등록하는 것을 권장한다.  @RequiredArgsConstructor에서 생성자를 해결해준다. 생성자를 직접 안쓰고 어노테이션을 쓰는 이유는 의존관계가 변경됨에 따라 수정해야하므로 이러한 작업을 줄이기 위해 어노테이션을 사용한다.  PostssSaveReqeuestDto  package com.arad4228Leaned.arad4228.web.dto;  import com.arad4228Leaned.arad4228.domain.posts.Posts; import lombok.Builder; import lombok.Getter; import lombok.NoArgsConstructor;  @Getter @NoArgsConstructor public class PostsSaveRequestDto {  private String title;  private String context;  private String author;  @Builder  public PostsSaveRequestDto(String title, String context, String author) {  this.title = title;  this.context = context;  this.author = author;  }   public Posts toEntity() {  return Posts.*builder*()  .title(title)  .context(context)  .author(author)  .build();  } }  이 클래스는 EntityClass와 거의 유사한 형태임에도 DtoClass를 추가로 생성 절대로 EntityClass를 RequestResponse클래스로 사용하면 안 된다.  EntityClass는 database와 맞닿은 핵심 클래스이기 때문이다.  PostsApiControllerTest  package com.arad4228Leaned.arad4228.web;  import com.arad4228Leaned.arad4228.domain.posts.Posts; import com.arad4228Leaned.arad4228.domain.posts.PostsRepository; import com.arad4228Leaned.arad4228.web.dto.PostsSaveRequestDto; import org.junit.jupiter.api.AfterEach; import org.junit.jupiter.api.Test; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.test.context.SpringBootTest; import org.springframework.boot.test.web.client.TestRestTemplate; import org.springframework.boot.web.server.LocalServerPort; import org.springframework.http.HttpStatus; import org.springframework.http.ResponseEntity;  import java.util.List;  import static org.assertj.core.api.AssertionsForClassTypes.*assertThat*;  @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.*RANDOM\_PORT*) public class PostApiControllerTest {   @LocalServerPort  private int port;   @Autowired  private TestRestTemplate restTemplate;   @Autowired  private PostsRepository postsRepository;   @AfterEach  public void tearDown() throws Exception {  postsRepository.deleteAll();  }   @Test  public void Posts\_등록된다() throws Exception {  // given  String title = "tile";  String context = "context";  PostsSaveRequestDto requestDto = PostsSaveRequestDto.*builder*()  .title(title)  .context(context)  .author("author")  .build();   String url = "http://localhost:" + port + "/api/v1/posts";   // when  ResponseEntity<Long> responseEntity = restTemplate.postForEntity(url, requestDto, Long.class);   // then  *assertThat*(responseEntity.getStatusCode()).  isEqualTo(HttpStatus.*OK*);  *assertThat*(responseEntity.getBody()).isGreaterThan(0L);   List<Posts> all = postsRepository.findAll();  *assertThat*(all.get(0).getTitle()).isEqualTo(title);  *assertThat*(all.get(0).getContext()).isEqualTo(context);  } } |  |
| * **특이사항 / 협업 사항**   3월 14일 전체 회의 전에도 Spring을 Gather를 통해 공부할 예정이다. | |